Attorney Docket No.: 16869B-025600US
Client Reference No.: HAL-ID 176

PATENT APPLICATION

VARIABLE ACCURACY MODES IN MICROPROCESSOR SIMULATION

Inventor: Sivaram Krishnan, a citizen of The United States, residing at
1723 Westbrook Avenue
Los Altos, CA 94024

Assignee: Hitachi America, Ltd.
New Marunouchi Bldg.
5-1, Marunouchi 1-Chome,
Chiyoda-ku
Tokyo, 100-8220

Entity: Large

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, 8th Floor
San Francisco, California 94111-3834
Tel: 415-576-0200

<u>PATENT</u>

Attorney Docket No.: 16869B-025600US
Client Reference No.: HAL-ID 176

VARIABLE ACCURACY MODES IN MICROPROCESSOR SIMULATION

CROSS-REFERENCES TO RELATED APPLICATIONS

[01] NOT APPLICABLE

FIELD OF THE INVENTION

[02] This invention relates generally to the field of computer software, and more particularly to a simulator used to predict system performance.

BACKGROUND OF THE INVENTION

[03] Simulators are often used to predict system (e.g., processor, memory, external modules, etc.) performance. Two common types of simulators used are functional simulators and performance (or cycle-accurate) simulators. Functional simulators simply emulate the semantic behavior of programs without giving information regarding execution times. On the other hand, performance simulators provide more detailed and accurate execution statistics including cycle times and detailed pipeline flows.

[04] However, due to the detailed nature of performance simulators they are somewhat slow. It is not uncommon for performance simulators to be 1,000 times slower than functional simulators. Often, a program may take days or weeks to run on a performance simulator. Consequently, detailed performance simulators may be useless for all but small programs. Moreover, performance simulators are not readily integrated into larger simulation environments as a result.

[05] One common use of simulators is analyzing the behavior of a small but critical code segment of a large program. The behavior of the remaining portions of the program is often irrelevant or of little relevance. It is often desired to gather statistics only for the small portion of code that is of interest. Since there is no easy mechanism for specifying the region of code of interest and limiting the gathering of statistics to that region alone, the statistics gathering is often polluted and takes longer than necessary.

SUMMARY OF THE INVENTION

[06] The present invention includes a simulator that supports multiple simulation modes, in one embodiment functional and performance simulation modes, and lets a user select between the two modes (i.e., analyze different portions of code differently) and vary the accuracy of the performance simulation.

[07] In a method according to one embodiment of the present invention, the performance of a system is simulated. The method comprises: performing simulation in a first simulation mode for at least a first portion of code that models at least a portion of the system; and performing simulation in a second simulation mode for at least a second portion of code that models at least a portion of the system.

[08] In a system for simulating the performance of a system according to another embodiment of the present invention, the system comprises: a module for performing simulation in a first simulation mode for at least a first portion of code that models at least a portion of the system; and a module for performing simulation in a second simulation mode for at least a second portion of code that models at least a portion of the system.

[09] A further understanding of the nature and advantages of the inventions herein may be realized by reference to the remaining portions of the specification and the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[10] Fig. 1 is a flow diagram according to an embodiment of the present invention;

[11] Fig. 2 is a flow diagram according to an embodiment of the present invention; and

[12] Fig. 3 illustrates subsystems of an exemplary computer system for use with the present invention.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

[13] As shown in the exemplary drawings wherein like reference numerals indicate like or corresponding elements among the figures, an embodiment of a system according to the present invention will now be described in detail. The following description sets forth an example of a simulator system and methodology. The system can be operated on many different computing platforms, and other variations should be apparent after review of this description.

[14] As mentioned previously, one embodiment according to the present invention provides a system that allows a user to perform both functional and performance simulation and allows the user to switch between the two modes as desired. This approach allows the user to run large programs quickly in the functional mode, while invoking the performance mode for various code regions of interest when it is important to examine cycle-accurate behavior. This system also allows the selective gathering of statistics. The user is given fine-grain control over the accumulation of statistics and the reported data is not polluted by data related to other portions of code.

[15] Generally, in one embodiment the basic environment is functional simulation and all programs are executed in this environment. At specific points of interest within the code representing the device being simulated, the user can invoke performance simulation. These points of interest can be indicated by the user (e.g., by using breakpoints, etc.). The accuracy of the performance simulation can be varied from one code portion to another code portion.

[16] In the performance simulation mode, the simulator provides system behavior information which may be cycle-accurate. The system can gather information related to processor pipelines, instruction and data buffers, branch predictions and penalties, cache and memory access times, etc. The performance simulation can be enabled and disabled at various points. These points include specific instruction addresses, data addresses, cycle counts, etc. By enabling and disabling the performance simulation at specific points of interest, e.g., procedure boundaries, it is possible to gather statistics for only that particular procedure(s). Additionally, by allowing initialization, clearing and printing of statistics at each of the desired points, it is possible to classify the data acquisition.

[17] In one embodiment, since the functional and performance simulation are logically and programmatically separate, a well-defined interface between the two modes is provided. It is desirable to keep the information passing through this interface as minimal as possible to reduce the data transfer cost. One exemplary embodiment of the interface can be a trace element represented by the following C data structure:

[18] struct trace_element

[19] {

[20] unsigned int ia;

[21] unsigned int opcode;

[22] unsigned int da;

[23] unsigned int misc;

[24] }

[25] In one embodiment, the functional simulation engine initializes the above data structure when performance simulation is initiated. It is the responsibility of the performance simulation engine to process the trace_element data.

[26] The performance simulation can be switched on or off at any regular breakpoint. Furthermore, finer grain control can be exercised over areas such as cache simulation, statistics gathering, etc. These aspects can be controlled via a graphical user interface (GUI), command line interface or the like.

[27] An exemplary set of some possible commands are as follows:

[28]	ta_start		- start performance simulation
[29]	ta_stop		- stop performance simulation
[30]	ta_set	icache	- enable icache simulation
[31]	ta_set	noicache	- disable icache simulation
[32]	ta_set	log	- print pipeline log
[33]	ta_set	stat	- gather statistics
[34]	ta_clear	stat	- clear statistics counters
[35]	ta_print	stat	- print statistics

[36] Referring to Fig. 1, in operation, at step S10 the simulator is launched. At step S12, the default state is set. Various settings can be made here, such as the default mode being functional simulation mode, etc.

[37] At step S14, user commands are accepted. For example, verification and test programs, applications, etc., can be loaded at this step. Commands to exercise and test a particular portion of a design can also be entered at this step. The type of simulation (performance or functional) can be set; i.e., changed from the default setting. Additionally, the number of instructions to be executed under a given mode (performance or functional) can be set at this point. Breakpoints (e.g., addresses, conditions, etc.) can also be set at this point so that the simulation returns to step S14 after the breakpoint is triggered. One can also turn on and off caches, etc. at this point.

[38] At step S16, the initial state is set (i.e., the simulator is initialized). At step S18, functional simulation begins. It should be understood that functional simulation can be considered a subset of performance simulation in one embodiment. That is, performance simulation includes functional simulation.

[39] At step S20, the simulation mode (functional or performance) is checked. This check can be done before or after or at the same time that functional

simulation begins. This check can be used to guard a number of instructions to execute in a given mode. For example, if at step S14 the user chooses to execute 100 instructions in functional simulation mode, then the mode check ensures that functional simulation only is performed for these 100 instructions. Likewise, if the user chooses to execute these 100 instructions in performance simulation mode, then this happens at step S26. However, since functional simulation can be considered to be a subset of performance simulation in one embodiment, functional simulation is also taking place.

[40] At step S22, if functional simulation mode was chosen, then functional statistics are computed and gathered for display, printing and/or saving. Some examples of the types of statistics computed and gathered are the number of instructions that were executed, the values of certain variables, how many times certain functions were called, the path taken and the dynamic execution sequences of instructions.

[41] At step S24, a check is made to see if the program (the application running on the simulator) has ended. If not, then the simulator returns to step S14 where user commands can again be accepted. For example, the user may now want to simulate 150 instructions in performance simulation mode with a first cache enabled and a second cache disabled.

[42] Turning now to Fig. 2, the simulator goes to step S50 if performance simulation was invoked (step S26 of Fig. 1). Basic performance simulation includes but is not limited to pipeline flows, functional unit usage, instruction dependency checks and branch penalties. At step S52, the program examines which caches, memories, etc. are to be considered (this might have been selected in step S14 of Fig. 1). At step S54, cache, etc., behavior and statistics are computed if appropriate.

[43] If not, then the program goes to step S56. This step can be done before, after or at the same time at step S52. At this point, the program determines which bus or busses are to be analyzed (this might have been selected in step S14 of Fig. 1). At step S56, bus behavior and statistics are computed if appropriate. At step S60, general information is computed and gathered which can be stored, printed and/or displayed. Next, the program returns to step S24 of Fig. 1. It is noteworthy that step combinations S52/S54 and S56/S58 are included for illustrative purposes. Any number of suitable step combinations that provide various functionality can be included in any suitable order.

[44] In one embodiment, the accuracy of the performance simulation can be altered as well. This can be in step S14 of Fig. 1. One must weigh the cost in simulation time against the benefit of increased accuracy. Thus, it has been shown that the user is able to select one or more of a plurality of code portions for performance simulation, in addition to the functional simulation, and at the same time adjust the accuracy of the performance simulation. The accuracy of the performance simulation can be adjusted for different code portions independently.

[45] It is noteworthy that the different modes (performance and functional simulation) can be invoked within a single simulation program execution run. Moreover, a given portion of code can be analyzed using one mode in one program run and using another mode in a distinct program run.

[46] Fig. 3 illustrates subsystems found in one exemplary computer system that can be used in accordance with embodiments of the present invention. Computers can be configured with many different hardware components and can be made in many dimensions and styles (e.g., laptop, palmtop, server, workstation and mainframe). Thus, any hardware platform suitable for performing the processing described herein is suitable for use with the present invention.

[47] Subsystems within are directly interfaced to an internal bus 210. The subsystems include input/output (I/O) controller 212, system random access memory (RAM) 214, central processing unit (CPU) 216, serial port 220, fixed disk 222 and network interface adapter 224. The use of the bus allows each of the subsystems to transfer data among the subsystems and, most importantly, with the CPU. External devices can communicate with the CPU or other subsystems via the bus by interfacing with a subsystem on the bus.

[48] Fig. 3 is illustrative of one suitable configuration for providing a system in accordance with the present invention. Subsystems, components or devices other than those shown in Fig. 3 can be added without deviating from the scope of the invention. A suitable computer system can also be achieved without using all of the subsystems shown in Fig. 3. Other subsystems such as a CD-ROM drive, graphics accelerator, *etc.*, can be included in the configuration without affecting the performance of the system included in the present invention.

[49] One embodiment according to the present invention is related to the use of an apparatus, such as the computer system, for implementing a simulator according to embodiments of the present invention. The processor 216 can execute one or more sequences of one or more instructions contained in the system memory 214. Such instructions may be read into memory 214 from a computer-readable medium, such as a fixed disk 222. Execution of the sequences of instructions contained in the memory 214 causes the processor to perform the process steps described herein. One or more processors in a multi-processing arrangement may also be employed to execute the sequences of instructions contained in the memory. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[50] The terms "computer-readable medium" and "computer-readable media" as used herein refer to any medium or media that participate in providing instructions to the processor 214 for execution. Such media can take many forms, including, but not limited to, non-volatile media, volatile media and transmission media. Non-volatile media include, for example, optical or magnetic disks, such as a fixed disk 222. Volatile media include dynamic memory, such as memory 214. Transmission media include coaxial cables, copper wire and fiber optics, including the wires that comprise the bus 210. Transmission media can also take the form of acoustic or light waves, such as those generated during radio frequency (RF) and infra-red (IR) data communications. Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, a hard disk, magnetic tape, any other magnetic medium, a CD-ROM disk, DVD, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, an EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave, or any other medium from which a computer can read.

[51] Various forms of computer-readable media may be involved in carrying one or more sequences of one or more instructions to processor 216 for execution. The bus carries the data to the memory 214, from which the processor retrieves and executes the instructions. The instructions received by the memory can optionally be stored on the fixed disk 222 either before or after execution by the processor.

[52] Many subsystem configurations are possible. Fig. 3 is illustrative of but one suitable configuration. Subsystems, components or devices other than those shown in Fig. 3 can be added. A suitable computer system can be achieved without using all of the subsystems shown in Fig. 3.

[53] The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. The scope of the invention should, therefore, be determined not with reference to the above description, but instead should be determined with reference to the appended claims along with their full scope of equivalents.